第 36 卷 第 9 期 计 算 机 工 程 2010 年 5 月
Vol.36 No.9 Computer Engineering May 2010
·软件技术与数据库· 文章编号：1000—3428(2010)09—0047—03 文献标识码：A 中图分类号：TP393

基于本体和多 Agent 的面向任务协同编程

符 丁，黄汉堂
(黔南民族师范学院计算机科学系，都匀 558000)

摘 要：针对面向对象编程与面向方面编程存在的问题，提出一种基于本体和多 Agent 的面向任务协同编程方法，给出任务分解原则、任务描述方法及编程的 9 个阶段。通过实例验证该方法可以提高目标软件系统的模块化程度，使目标软件系统易于实现、理解、演化和复用。
关键词：面向对象编程；面向方面编程；面向任务协同编程

Task-oriented Cooperative Programming Based on Ontology and Multi-Agent

FU Ding, HUANG Han-tang
(Department of Computer Science, Qiannan Normal College for Nationlities, Duyun 558000)

【Abstract】The task-oriented cooperative programming based on Ontology and multi-Agent is put forward aiming at the drawback of the Object-Oriented Programming(OOP) and Aspect-Oriented Programming(AOP). This paper introduces the decomposition principle and descriptive method of task, and nine programming steps. An example which shows how to apply the method is given. It improves the modularity of crosscutting concerns in software, so that software is easily realizable, readable, evolutionary and reusable.
【Key words】Object-Oriented Programming(OOP); Aspect-Oriented Programming(AOP); task-oriented cooperative programming

1 概述

一个典型的软件系统由许多核心级关注点和系统级关注点组成。核心级关注点是系统要完成的业务功能；而系统级关注点是完成核心级关注点所必需的配套设施，这些配套设施通常被认为是整个系统特性，或者是业务功能的功能约束。面向对象编程(Object-Oriented Programming, OOP)成功地实现了核心关注点的模块化，但难以模块化处理软件系统的横切关注点，以至于横切关注点的代码散布于若干个核心关注点代码中，使核心关注点的代码和横切关注点的代码相互交织，最终使得目标系统难以实现、理解、演化和复用。研究人员开始寻找一种能模块化处理软件系统的横切关注点的技术，最终由 Gregor Kiczales 等在 1997 年欧洲面向对象编程大会上提出了面向方面的编程(Aspect-Oriented Programming, AOP)。面向方面的编程解决了横切关注点的模块化问题，但 AOP 本身还有许多问题有待于进一步研究，如大量 Aspect 怎样协同工作，它们怎样组合[1]。针对面向对象编程与面向方面编程存在的问题，笔者在经过多年软件开发实践和理论研究的基础上提出了一种基于本体和多 Agent 的面向任务协同编程(简称面向任务协同编程)，主要解决 3 个问题：(1)提高目标软件系统中贯穿特性的模块化程度；(2)开发团队高效协同地完成项目开发；(3)软件实体具有较好的动态和复杂的协同工作能力。

2 基于Ontology和多Agent的协同编程

2.1 Agent 技术

软件 Agent 是指运行于计算机上为用户完成有用任务的软件实体(对象)，它具有交互协作性、自主性、可控性和目标任务驱动性。Agent 与 Object 相比，是一种比较高级的、粗粒度的、可重用、可包含心智状态(BDI)的对象，能通过与外界(包括其他 Agent 及其拥有者)的通信进行感知，并根据感知结果(外部事件)及内部状态的变化独立决定和控制自身行为的相对独立的综合实体[2]。多 Agent 系统(Multi-Agent System, MAS)是由多个 Agent 组成的集合，各 Agent 之间的活动是自治和独立的，行为和意图不受其他 Agent 限制，它们之间通过合作、协商、竞争、交互协作共同完成系统设定的目标(任务)。引入 Agent 技术的目的是提高目标软件系统中贯穿特性的模块化程度，从而使目标软件系统易于实现、理解、演化，提高软件的可复用性。

2.2 Ontology

Ontology 是描述概念及概念之间关系的概念模型。Robert Neches 等人给的定义是，“Ontology 是关于相关领域基础术语和相互关系词汇表的定义以及使用这些术语和相互关系定义扩展词汇表的规则”。Ontology 的目标是捕获相关领域的知识，提供对该领域知识的共同理解，确定该领域内共同认可的词汇，并从不同层次的形式化模式上给出这些词汇(术语)和词汇间相互关系的明确定义。引入 Ontology 是对面向任务协同编程和待开发目标软件系统领域的有关概念及其关系给予明确清晰的定义，其目的有 2 个：(1)有助于开发团队成员协作顺利完成目标软件系统的开发；(2)从另一个方面使目标软件系统易于实现、理解、演化和复用。下面介绍面向任务协同编程的部分概念。

(1)Task：任务指通过 Agent 执行某一活动并达到某一目

作者简介：符 丁(1964－)，男，讲师、硕士，主研方向：软件工程，面向对象编程；黄汉堂，讲师
收稿日期：2010-01-04 **E-mail：**hnhbwhfd@126.com

的。复杂任务可以分解成为简单子任务，子任务由一个 Agent 完成(即模块化)。此时的子任务称作单元任务，这些子任务相互关联又相互独立。子任务相互关联指需要多个 Agent(多个角色)相互协作才能完成父任务，任务执行具有时序性，子任务的独立性指只需一个 Agent 就可以完成任务(即模块化)。这种任务也可由 Agent 分成几个阶段完成，即此任务由多个阶段性任务组成，若干阶段性任务分别由 Agent 中若干个不同服务完成。阶段性任务又称为基本任务，单元任务通过执行一个或多个基本任务完成。任务实现是按照一定顺序完成的，任务开始由其他任务完成或者事件触发。

(2)Cooperation：协作是多个同步或异步协作活动的交织过程。协作以任务为中心，是在情景上下文中动态生成的。

(3)Object：对象指一个物理或概念性的实体，物理实体对象如用户，概念实体如消息。对象包含属性和其他的属性值，对象由某一个任务执行某类动作而改变状态。任务执行时对象被调用，对象通过引发事件发生而影响任务执行顺序。

(4)Agent：代理是任务执行主体，通常是指人或能完成一定功能的软件，代理并不具体到某一个体，而是指一类具有某一特征或能力的个体，代理包含一定技能并且总是代表某类角色执行任务，一个代理可以扮演多类角色。

(5)Role：角色是关于某一类权限及职责的集合。

(6)Event：事件指协同过程中某时间点上状态的改变，事件通过触发任务影响任务的执行顺序[3]。

2.3 面向任务协同编程技术的基本思想

一个待开发的目标系统具有多视角，从用户角度看，目标软件系统是一个提供服务的系统(即服务)；从开发人员的角度看，目标软件系统是一个待求解的问题；从程序的角度看，目标软件系统是一项由许多个对象(软件实体)相互协作共同完成的任务。

面向任务协同编程的基本思想是：将待开发的目标系统看作一项任务，假设这个系统(或任务)很复杂或者很大，表明它分别包含若干个核心级和系统级关注点，不能够被模块化，可以将该系统(或任务)按照某种合理、自然、科学的分离技术(如按其提供的服务功能分解、按领域分解)分解成若干个子系统(或子任务)，如果子系统(或子任务)仍然比较复杂，再对子系统按照某种科学的方法进行分解，……，如此下去，直到每个子系统(或子任务)只含一个关注点，能够被模块化为止。系统或任务经过多次分解产生一个多层树形结构，树的叶子就是不能再分割的子任务，该子任务可以由一个对象(Agent)完成。那么整个任务在协同机制(协调与调度 Agent)的作用下，由若干个对象(Agent)相互协作共同完成。

2.4 面向任务协同编程的主要阶段

面向任务协同编程主要分为 9 个阶段：

(1)任务分解，目标软件系统视作一项任务，根据任务分解原则和分解方法对其进行分解，任务分解的制品是任务分解图和任务流程图。任务分解图反映了父任务和子任务之间的组合关系；任务流程图描述了组成父任务的子任务之间的时序和逻辑关系。

(2)定义角色，角色负责实现系统的单元任务，具有原子性。角色范围从 4 个属性进行定义：权限，职责，行为和协议。这个阶段产生角色模型。

(3)确定 Agent，即为 Agent 分配角色，这个阶段产生 Agent 类图和交互图(表示 Agent 之间交互)。

(4)构建对话，对话定义了 2 个 Agent 之间的协作协议，该阶段产生对话图。

(5)组装 Agent，即定义 Agent 的架构和定义构架的组建。该阶段产生 Agent 架构图。

(6)系统设计，该阶段使用部署图展示一个系统中 Agent 实例的数目、类型和位置。

(7)系统实现。

(8)系统测试。

(9)系统投入运行。

2.5 任务分解原则、方法和描述方法

2.5.1 任务分解原则

面向任务协同编程时，任务分解应遵循以下原则：

(1)独立性：所划分的任务要具有一定的独立性，这样有助于各对象独立处理各任务，减少相互间协调通信的工作；子任务的目标要非常明确，相对独立。某一个子任务不一定要知道其他子任务是什么及如何实现，只要知道其自身要完成什么工作，如何完成这一工作以及完成该工作所需的信息。

(2)层次性：一个任务可分解为多个子任务，子任务又可分解为多个下层子任务，复杂任务可分解为多个简单的、易于处理的任务。

(3)模块化：分解后的最终子任务只能由一个对象完成，子任务之间没有包含关系、从属关系和上下层关系。

(4)领域化：分解后的子任务不能含有 2 个或 2 个以上的领域任务(问题)，这样有助于专业人员解决属于自己领域的任务(问题)。

(5)适度性：任务的大小、个数和层次要适中。

2.5.2 任务分解技术

在面向任务协同编程时，任务分解不仅要遵循任务分解原则，还需要有合理、自然、科学的任务分解方法。

(1)服务功能法：按目标系统提供的服务功能对其进行分解。对用户而言，目标系统提供的是服务。通常一个目标系统提供的服务是多项的，即目标系统具有多个服务功能。

(2)问题领域法：系统是由不同的领域任务(问题)组成任务(问题)(即系统是复合任务或问题)，则将系统按领域分解，分解后的子系统(子任务或子问题)属于同一个领域。这样有助于领域专业人员并行独立解决子任务或子问题。

(3)贯穿特性法：将系统中具有贯穿特性的关注点分离出来，有助于各关注点模块化。

(4)专业化方法：将任务所属专业按该专业特有分解方法进行分解。

2.5.3 任务分解描述方法

一个任务的结构有静态结构和动态结构，静态结构说明了父任务和子任务之间的组合关系，而动态结构描述了组成父任务的子任务之间的顺序和逻辑关系。为了详细揭示父任务与子任务之间、子任务与子任务之间的结构关系，需要采用恰当的方法予以描述。

(1)任务分解结构表示法

任务是整个协同活动的基本目标，复杂的任务可以分解成简单的子任务，任务分解的实质是目标的分解，即大的目标变成一系列小的目标，反映大的目标与子目标之间的组合关系(这是一种静态关系)。任务分解图能表示这种组合关系。它是一种扩展的与或关系，不具有条件与和条件或关系。

在图 1 所示任务分解图中，任务用方框图表示，其名称放在方框内，任务分解如果是无条件分解，用实箭线表示，如果是有约束分解(指子任务有条件执行)，用虚箭线表示，

与其他子任务构成条件与和条件或关系；所有箭头都由父任务指向子任务，表示层次关系；箭头线之间若用圆弧连接表示多个子任务之间具有与关系，必须全部完成后才能完成父任务，否则，子任务之间是或关系，只要其中完成一个以上子任务就能完成父任务。

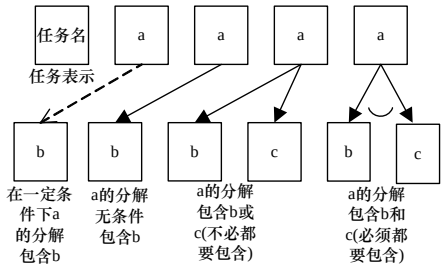

图 1　任务分解图元素

任务分解图是一种具有层次结构的与或图，称为与或树。根节点表示一个总任务；中间节点一方面是上层节点(父任务)分解所得的子任务，另一方面是下层节点的父任务；叶节点表示构成总任务的单元任务。单元任务一般只包含一个关注点，能够模块化(即任务由一个对象完成)。

(2)任务流程结构表示方法

任务分解图可以描述任务的静态结构，但是不能表现子任务之间的时序和逻辑关系，即任务的动态结构。任务流程图用于描述任务的动态结构，类似程序流程图，但是，对程序流程图的表示方法进行了扩展，借用了 UML 活动图的同步并发描述机制。在任务流程图中，方框表示子任务，方框内的子任务名带有隶属符号“.”，用于指出子任务的执行主体——角色。在框内右上角，用符号“*”表示子任务重复执行多次；用菱形表示子任务的执行条件；用箭头线表示顺序关系；分劈和结合表示任务的并发和同步，分劈同步杆有一条转入箭头线和多条转出箭线，每条箭线代表一个任务，多个任务并行执行；结合同步杆有多条转入箭线和一条转出箭线，表示多个并发执行的任务都完成后才能执行一个新的任务，见图 2。

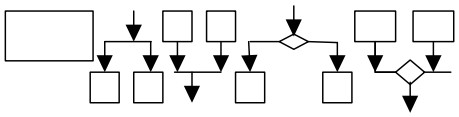

图 2　任务流程图元素

任务流程图能够描述整体任务的动态结构，但是要求子任务必须是单元任务或基本任务，即子任务之间没有包含、重叠关系，语义上各自独立，并且由一个角色独立完成，否则，任务流程图就描述不了该整体任务的动态结构[4]。

2.6　实例说明

以下用简化的银行储蓄系统为实例阐明面向任务协同编程的基本思想和阶段。假设简化的银行储蓄系统只提供存款和取款 2 项服务。将储蓄系统按提供的服务功能分解为存款子系统(子任务)和取款子系统(子任务)，存款子系统包含日志、用户身份鉴别、持久存储和存款 4 个关注点；取款子系统包含日志、用户身份鉴别、安全、持久存储和取款 5 个关注点。再按领域分离技术对存款子系统(子任务)和取款子系统(子任务)进行分解，分解所形成的树形结构储蓄系统任务分解图如图 3 所示。

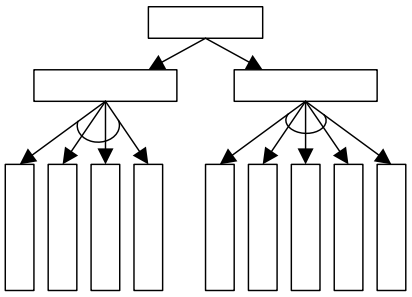

图 3　储蓄系统任务分解图

在树形结构储蓄系统任务分解图中，日志、用户身份鉴别、持久存储、安全、存款、取款 6 个关注点可分别单独模块化且具有稳定性，无须再对其进行分解。存款和取款是核心级关注点，而日志、用户身份鉴别、持久存储、安全是系统级关注点。每个核心级关注点和每个系统级关注点均可分配一个 Agent 来实现(即日志任务由处理日志的 Agent 完成，用户身份鉴别的任务由鉴别用户身份的 Agent 完成，持久存储的任务由负责持久存储的 Agent 完成，安全的任务由负责安全的 Agent 完成，存款和取款的任务分别由负责存款和负责取款的 Agen 完成)。这些 Agent 在协同机制(协调与调度 Agent)的作用下，相互协作共同完成存款和取款的任务。协调与调度 Agent 的作用：一方面进行任务分配(即给相应的 Agent 下达任务通知书)；另一方面根据任务流程图协调控制 Agent 执行任务的顺序[5]。

图 4 是 Agent 组织三层模式。

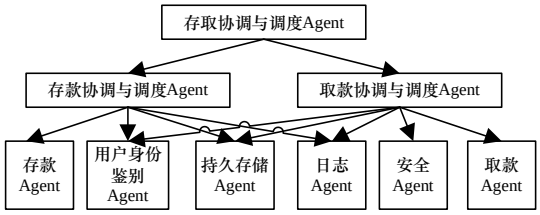

图 4　Agent 组织三层模式

3　结束语

本文总结了笔者为某金融公司主持开发一个综合业务系统的经验。采用基于 Ontology 和多 Agent 的面向任务协同编程方法主要是基于以下 3 点考虑：(1)系统较大，开发人员 36 人，如何高效协作完成该项目开发。(2)如何满足业务需求的动态性(即系统的演化)。(3)系统只作小部分增减就可应用于其他金融公司相关业务系统(即系统的复用)。经多年实践证明该方法达到了预期的效果。

参考文献

[1] 曹东刚, 梅　宏. 面向 Aspect 的编程——一种新的编程范型[J]. 计算机科学, 2003, 30(9): 5-10.

[2] 韦东方, 薛恒新, 游　专. 任务/目标驱动的 ERP 系统构建的分析与应用[J]. 计算机应用, 2004, 24(11): 56-59.

[3] 龚　能, 李玉顺, 史美林. 协作环境中的关键技术研究[J]. 计算机科学, 2005, 32(9): 230-233.

[4] 曹裕华, 冯书兴, 徐雪峰. 作战任务分解的概念表示方法研究[J]. 计算机仿真, 2007, 24(8): 1-4.

[5] 李宪港, 李中学. 基于使命的 Agent 组织模型[J]. 计算机工程, 2008, 34(23): 187-189.

编辑　张正兴